

# Süsteemprogrammeerimine keeles C



Mälu mõistmine I  
Standardsisend

# Konstandid

- ♦ 33 – (märgiga) täisarv
- ♦ 33u – märgita täisarv
- ♦ 33l – long int tüüpi täisarv
- ♦ 33lu – märgita long int tüüpi täisarv
- ♦ 033 – täisarv kaheksandsüsteemis
- ♦ 0x33 – kuueteistkümnendsüsteemis

# Ujukomaarvud

- ♦ Double on traditsiooniline; ANSI puhul float
- ♦ 33f, 33F, 33.0
- ♦ 33e1 – eksponentkujul esitatult
- ♦ 33E1
- ♦ 33.0L – long double

# Char

- ♦ ASCII tähemärkide jaoks on 'x'
- ♦ \t – tab
- ♦ \n – reavahetus
- ♦ \b – piiks/plänn/müra
- ♦ \v – vertikaalne tab
- ♦ \' – '
- ♦ \" – ''
- ♦ \\ – \
- ♦ \0xff – ASCII märk väärtusega 255

# Memory

- Processor registers (hidden in C)
- RAM
- Devices (hard disk)
- Internet?
- People??
- Books???

# Stack

- Simple data structure
- Efficient implementations
- FILO (as opposed to FIFO)
- Operations: Push, Pop
- Important to us due to call stack
- Often supported in hardware

# C implementation

```
typedef struct {  
    int size;  
    int items[STACKSIZE];  
} STACK;
```

```
void push(STACK *ps, int x)  
{  
    if (ps->size == STACKSIZE) {  
        fputs("Error: stack  
overflow\n", stderr);  
        abort();  
    } else  
        ps->items[ps->size++] = x;  
}
```

```
int pop(STACK *ps)  
{  
    if (ps->size == 0){  
        fputs("Error: stack  
underflow\n", stderr);  
        abort();  
    } else  
        return ps->items[--ps->size];  
}
```

# Hardware implementation

- Special stack register (can be read/written)
  - We will name it %esp in further examples
- Assembly instructions to manipulate it



# The Dreadful Assembly

# Short introduction to assembly

- Mainly moves data around (mov series)
- Jumps, conditional jumps (jmp series)
- Arithmetics
- Management (push, pop, call, return)
- Examples from IA32
  - Word = 16 bit due to ancient history
  - Double word for 32 bits

# Registers

- 8 registers for 32bit values
- General purpose: %eax %ecx %edx %ebx %edi %esi (Historical names, would be simpler)
- Fun registers: %esp %ebp (Stack pointer & Frame pointer)
- Can be addressed also in smaller segments
- %eax[            %ax[%ah[   ] %al[   ] ]

# Operands

- Instructions have operands (arguments)
- Immediate
  - Constant values
  - \$1024, \$-10, \$0xdeadbeef
- Register
  - %eax, %al
- Memory
  - $24(\%eax, \%edx, 1) \sim Immediate(reg_b, reg_i, scale)$

# Examples

```
movl $0x5040, %eax  
movl %ebp, %esp  
movl (%edi, %ecx), %eax  
movl $-17, (%esp)
```

# Stack operations

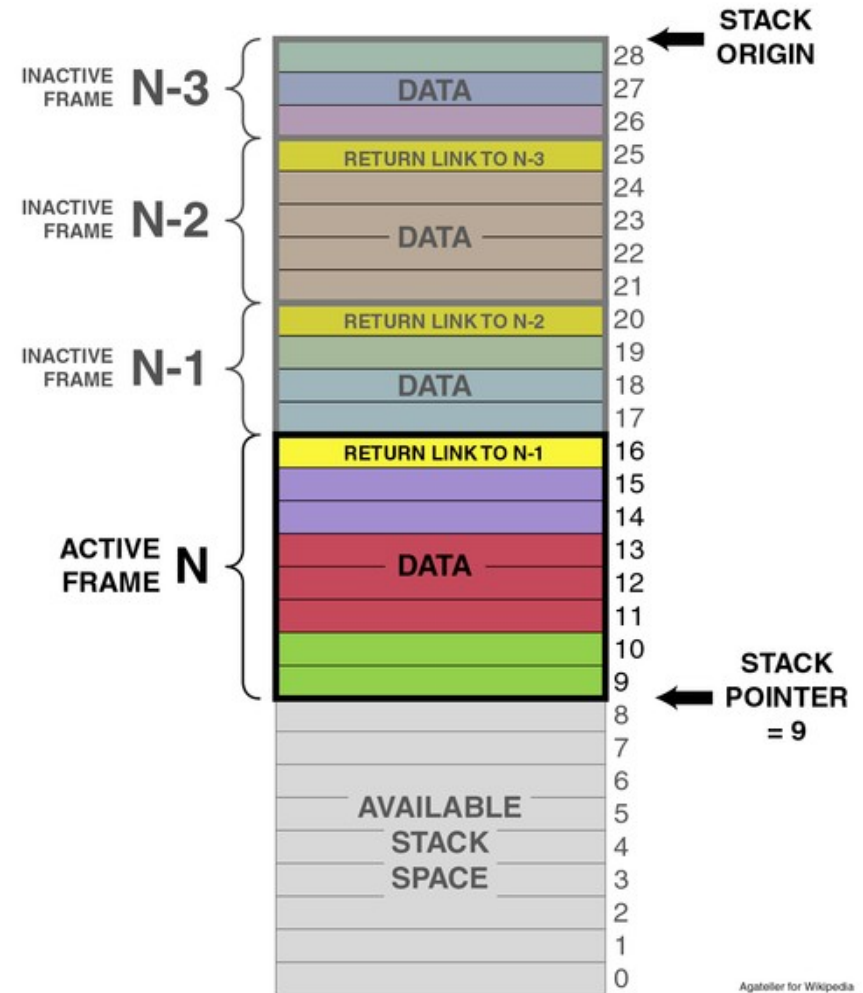
- $\%eax = 0x123$        $\%edx = 0$        $\%esp = 0x108$
- `pushl %eax`
- $\%eax = 0x123$        $\%edx = 0$        $\%esp = 0x104$
- `popl %edx`
- $\%eax = 0x123$        $\%edx = 123$        $\%esp = 0x108$

# Procedures

- Call involves passing both data and control from one code to another.
- Must store local variables and arguments, deallocate them on exit

# Stack frame

- Uses frame pointer to keep track of previous frame
- Stack pointer tracks “top” of stack





# One frame contains

- Address of last %ebp
  - Current frame pointer points to it (data accessed in relation to it)
- Saved registers
- Local variables ( out of registers; array; & )
- Any temporary data
- Argument building area
- Return address (only if not active frame)

# Transfer of control

- For procedure calls, processor supports the following instructions:
  - **call *Label* / call \**Operand*** – calls procedure
  - **leave** – prepares stack for return
  - **ret** – return from call

1. Prepare stack
2. call procedure
3. Profit

# Call instruction

- Can start executing from an address or a label
- Pushes return address to stack (return address is next instruction from the call)
- Jumps to called address (= set program counter to the start of a procedure)

# Ret instruction

- Pop an address from stack
- Go to the address with program counter
- *To use properly, stack pointer must point to the “bookmark” address that call instruction stored.*
- For preparation, leave instruction is used

Leave:

```
movl %ebp, %esp
```

```
popl %ebp
```

```
# note: %ebp == stack frame
```

# Recap

- *call* pushes return address to stack, jumps
- new procedure saves old stack frame to stack
- Copies current stack pointer to frame pointer
- ...
- Copy frame pointer to stack pointer
- Restore old frame pointer
- Return to stored bookmark

# Register conventions

- `%eax`, `%edx`, `%ecx` – Caller save
  - Procedures can overwrite them as want, but must restore them after return, as they may get overwritten
- `%ebx`, `%esi`, `%edi` – Callee save
  - Procedures can overwrite them only if they save them and restore them before returning
- `%eax` is the return register

# What reflects to C?

- Automatic variables live on stack
- Function arguments are copied to stack before calling (call by value)
- Using pointers as arguments to functions can make calls by reference
- Uninitialized variables contain garbage
- Pointers to freed stack contain garbage
- Writing over a stack frame pointer is Not Good
- Writing over the return address is worse

# Buffer overflow exploitation

- When a buffer overflows, it is possible to write over the return pointer to point within the buffer itself
- The buffer gets executed



# How to remove variable from stack?

- Easy, declare it as *static*.
  - `static int i = 0xf00;`
- Moves the variable to heap

# Standardsisend ja väljund

- Unixi omaaegne tugev külg
- “Kõik on kujutatav failina”
  - Seadmed ( /dev : klaviatuur, helikaart, võrgukaart)
  - Protsessid ( /proc kataloog )
  - Kerneli olek ( /sys kataloog )
  - Muuhulgas ka programmide sisendid-väljundid

# getchar() ja putchar()

- Lihtsuse huvides teeme praegu katsed getchar() ja putchar() abil. Hiljem veidi rohkem.
- Loevad/kirjutavad (puhverdatult) ühe tähemärgi
- Faili lõpp EOF (väärtusega -1) tähendab, et peab getchar-i küsima int andmetüübina
- Reapuhvri tõttu ei jõua sisend enne reavahetust programmini,
  - Seda on saame soovil välja lülitada
- Ka väljund on nii puhverdatud

# Sisendi ümbersuunamine

- Olulised märgid on < > ja |
- cat < failinimi – näitab faili välja
- cat > failinimi – kirjutab faili
- cat < üks > kaks – kopeerib faili üks nime alla kaks
- ls | more – esitab käsu ls väljundi lehekülgedena
- cat viganenimi 2> err.log – saadab vead faili